OpenLDAP 2.0 Administrator's Guide

The OpenLDAP Project <http://www.openldap.org/>
15 September 2000

Table of Contents

Preface

Copyright

Scope of this Document

This document provides a guide for installing OpenLDAP 2.0 Software on UNIX (and UNIX-like) systems. The document is aimed at experienced system administrators but who may not have prior experience operating LDAP-based directory software.

This document is meant to be used in conjunction with other OpenLDAP information resources provided with the software package and on the project's extensive site (http://www.OpenLDAP.org/) on the World Wide Web. The site makes available a number of resources.

OpenLDAP Resources

Resource	URL
Document Catalog	http://www.OpenLDAP.org/doc/
Frequently Asked Questions	http://www.OpenLDAP.org/faq/
Issue Tracking System	http://www.OpenLDAP.org/its/
Mailing Lists	http://www.OpenLDAP.org/lists/
Software Pages	http://www.OpenLDAP.org/software/
Support Pages	http://www.OpenLDAP.org/support/

Acknowledgments

The OpenLDAP Project is comprised of a team of volunteers. This document would not be possible without their contribution of time and energy.

The OpenLDAP Project would also like to thank the University of Michigan LDAP for building the foundation of LDAP software and information to which OpenLDAP Software is built upon.

Amendments

Suggested enhancements and corrections to this document should be submitted using the OpenLDAP *Issue Tracking System* (http://www.openldap.org/its/).

About this document

This document was produced using the *Simple Document Format* (http://www.mincom.com/mtr/sdf/) documentation system developed by *Ian Clatworthy*.

---

1. Introduction to OpenLDAP Directory Services

This document describes how to build, configure, and operate OpenLDAP software to provide directory services. This includes details on how to configure and run the stand-alone LDAP daemon, *slapd*(8) and the stand-alone LDAP update replication daemon, *slurpd*(8). It is intended for newcomers and experienced administrators alike. This section provides a basic introduction to directory services and, in particular, the directory services provided by *slapd*(8).

1.1. What is a directory service?

A directory is specialized database optimized for reading, browsing and searching. Directories tend to contain descriptive, attribute-based information and support sophisticated filtering capabilities. Directories generally do not support complicated transaction or roll-back schemes found in database management systems designed for handling high-volume complex updates. Directory updates are typically simple all-or-nothing changes, if they are allowed at all. Directories are tuned to give quick-response to

high-volume lookup or search operations. They may have the ability to replicate information widely in order to increase availability and reliability, while reducing response time. When directory information is replicated, temporary inconsistencies between the replicas may be okay, as long as they get in sync eventually.

There are many different ways to provide a directory service. Different methods allow different kinds of information to be stored in the directory, place different requirements on how that information can be referenced, queried and updated, how it is protected from unauthorized access, etc. Some directory services are *local*, providing service to a restricted context (e.g., the finger service on a single machine). Other services are global, providing service to a much broader context (e.g., the entire Internet). Global services are usually *distributed*, meaning that the data they contain is spread across many machines, all of which cooperate to provide the directory service. Typically a global service defines a uniform *namespace* which gives the same view of the data no matter where you are in relation to the data itself. The Internet Domain Name System is an example of a globally distributed directory service.

1.2. What is LDAP?

slapd's model for directory service is based on a global directory model called LDAP. LDAP stands for Lightweight Directory Access Protocol. LDAP is a directory access protocol that runs over TCP/IP. The nitty-gritty details of LDAP are defined in RFC2251 "The Lightweight Directory Access Protocol (v3)." This section gives an overview of LDAP from a user's perspective.

What kind of information can be stored in the directory? The LDAP information model is based on *entries*. An entry is a collection of attributes that has a globally-unique Distinguished Name (DN). The DN is used to refer to the entry unambiguously. Each of the entry's attributes has a *type* and one or more *values*. The types are typically mnemonic strings, like "`cn`" for common name, or "`mail`" for email address. The syntax of values depend on the attribute type is. For example, `cn` attribute might be the value `Babs Jensen`. A `mail` attribute might contain the value "`babs@example.com`". A `jpegPhoto` attribute would contain a photograph in the JPEG (binary) format.

How is the information arranged? In LDAP, directory entries are arranged in a hierarchical tree-like structure. Traditionally, this structure reflected the geographic and/or organizational boundaries. Entries representing countries appeared at the top of the tree. Below them are entries representing states and national organizations. Below them might be entries representing organizational units, people, printers, documents, or just about anything else you can think of. Figure 1.1 shows an example LDAP directory tree using traditional naming.

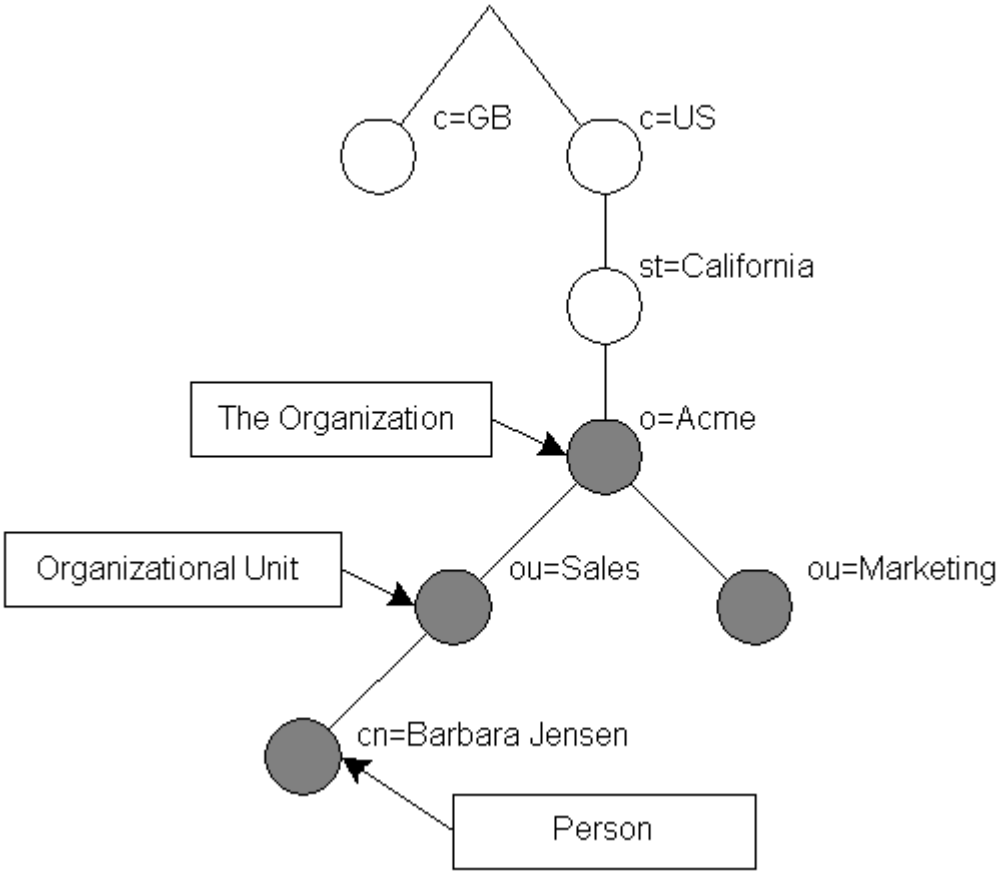

Figure 1.1: LDAP directory tree (traditional naming)

The tree may also be arranged based upon Internet domain names. This naming approach is becoming increasing popular as it allows for directory services to be locating using the Domain Name System. Figure 1.2 shows an example LDAP directory tree using domain-based naming.

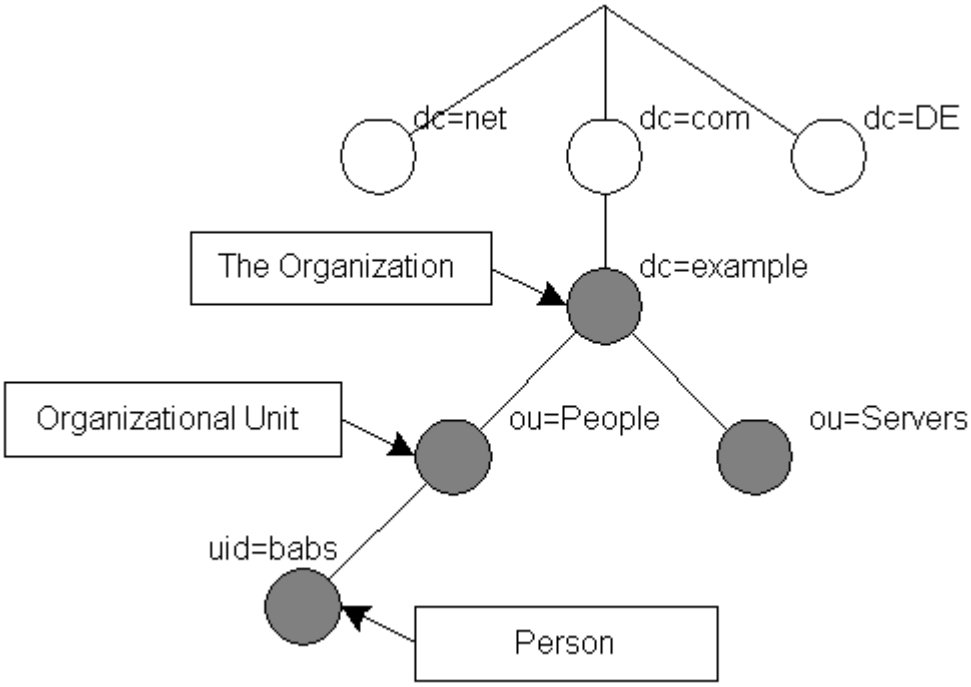

Figure 1.2: LDAP directory tree (Internet naming)

In addition, LDAP allows you to control which attributes are required and allowed in an entry through the use of a special attribute called `objectClass`. The values of the `objectClass` attribute determine the *schema* rules the entry must obey.

How is the information referenced? An entry is referenced by its distinguished name, which is constructed by taking the name of the entry itself (called the Relative Distinguished Name or RDN) and concatenating the names of its ancestor entries. For example, the entry for Barbara Jensen in the Internet naming example above has an RDN of `uid=babs` and a DN of `uid=babs,ou=People,dc=example,dc=com`". The full DN format is described in RFC2253, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names."

How is the information accessed? LDAP defines operations for interrogating and updating the directory. Operations are provided for adding and deleting an entry from the directory, changing an existing entry, and changing the name of an entry. Most of the time, though, LDAP is used to search for information in the directory. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter. Information can be requested from each entry that matches the criteria.

For example, you might want to search the entire directory subtree at and below `dc=example,dc=com` for people with the name `Barbara Jensen`, retrieving the email address of each entry found. LDAP lets you do this easily. Or you might want to search the entries directly below the `st=California,c=US` entry for organizations with the string `Acme` in their name, and that have a fax number. LDAP lets you do this too. The next section describes in more detail what you can do with LDAP and how it might be useful to you.

How is the information protected from unauthorized access? Some directory services provide no protection, allowing anyone to see the information. LDAP provides a mechanisms for a client to authenticate, or prove its identity to a directory server, paving the way for rich access control to protect

the information the server contains. LDAP also supports privacy and integrity security services.

1.3. How does LDAP work?

LDAP directory service is based on a *client-server* model. One or more LDAP servers contain the data making up the LDAP directory tree. An LDAP client connects to an LDAP server and asks it a question. The server responds with the answer and/or with a pointer to where the client can get additional information (typically, another LDAP server). No matter which LDAP server a client connects to, it sees the same view of the directory; a name presented to one LDAP server references the same entry it would at another LDAP server. This is an important feature of a global directory service, like LDAP.

1.4. What is slapd and what can it do?

slapd is an LDAP directory server that runs on many different platforms. You can use it to provide a directory service of your very own. Your directory can contain pretty much anything you want to put in it. You can connect it to the global LDAP directory service, or run a service all by yourself. Some of slapd's more interesting features and capabilities include:

LDAPv2 and **LDAPv3**: *slapd* supports both version 2 and 3 of the Lightweight Directory Access Protocol. *slapd* provides support for the latest features while maintaining interoperability with existing clients. *slapd* supports both IPv4 and IPv6.

Simple Authentication and Security Layer: *slapd* supports strong authentication services through the use of SASL. *slapd*'s SASL implementation utilizes Cyrus SASL software which supports a number of mechanisms including DIGEST-MD5, EXTERNAL, and GSSAPI.

Transport Layer Security: *slapd* provides privacy and integrity protections through the use of TLS (or SSL). *slapd*'s TLS implementation utilizes OpenSSL software.

Access control: *slapd* provides a rich and powerful access control facility, allowing you to control access to the information in your database(s). You can control access to entries based on LDAP authorization information, IP address, domain name and other criteria. *slapd* supports both *static* and *dynamic* access control information.

Internationalization: *slapd* supports Unicode and language tags.

Choice of databases: *slapd* comes with a variety of different backend databases you can choose from. They include LDBM, a high-performance disk-based embedded database; SHELL, a database interface to arbitrary shell scripts; and PASSWD, a simple password file database. LDBM utilizes either BerkeleyDB or GDBM.

Multiple database instances: *slapd* can be configured to serve multiple databases at the same time. This means that a single *slapd* server can respond to requests for many logically different portions of the LDAP tree, using the same or different backend databases.

Generic modules API: If you require even more customization, *slapd* lets you write your own modules easily. *slapd* consists of two distinct parts: a front end that handles protocol communication with LDAP clients; and modules which handle specific tasks such as database operations. Because these two pieces

communicate via a well-defined C API, you can write your own customized modules which extend *slapd* in numerous ways. Also, a number of *programmable database* modules are provided. These allow you to expose external data sources to *slapd* using popular programming languages (Perl, *Shell*, SQL, and TCL).

Threads: *slapd* is threaded for high performance. A single multi-threaded *slapd* process handles all incoming requests, reducing the amount of system overhead required.

Replication: *slapd* can be configured to maintain replica copies of its database. This *single-master/multiple-slave* replication scheme is vital in high-volume environments where a single *slapd* just doesn't provide the necessary availability or reliability. *slapd* also includes experimental support for *multi-master* replication.

Configuration: *slapd* is highly configurable through a single configuration file which allows you to change just about everything you'd ever want to change. Configuration options have reasonable defaults, making your job much easier.

slapd also has its limitations, of course. The main LDBM database backend does not handle range queries or negation queries very well. These features and more will be coming in a future release.

1.5. What about X.500?

Technically, LDAP is a directory access protocol to an X.500 directory service, the OSI directory service. Initial LDAP servers were gateways between LDAP and the X.500 Directory Access Protocol (DAP). DAP is a heavyweight protocol that operates over a full OSI protocol stack and requires a significant amount of computing resources. LDAP is designed to operate over TCP/IP and provides most of the functionality of DAP at a much lower cost.

This use of LDAP makes it easy to access the X.500 directory, but still requires a full X.500 service to make data available to the many LDAP clients being developed. As with full X.500 DAP clients, a full X.500 DAP server is no small piece of software to operate.

The stand-alone LDAP daemon, or *slapd*(8), is meant to remove much of the burden from the server side just as LDAP itself removed much of the burden from clients. If you are already running a X.500 DAP service and you want to continue to do so, you can probably stop reading this guide, which is all about running LDAP via *slapd*, without running X.500 DAP. If you are not running X.500 DAP, want to stop running X.500 DAP, or have no immediate plans to run X.500 DAP, read on.

It is possible to replicate data from an LDAP directory server to a X.500 DAP DSA. This requires an LDAP/DAP gateway. OpenLDAP does not provide such a gateway, but our replication daemon can be used to replicate to such a gateway. See the Replication with slurpd chapter of this document for information regarding replication.

1.6. What is slurpd and what can it do?

slurpd(8) is a daemon that helps *slapd* provide replicated service. It is responsible for distributing changes made to the master *slapd* database out to the various *slapd* replicas. It frees *slapd* from having to worry that some replicas might be down or unreachable when a change comes through; *slurpd* handles retrying failed requests automatically. *slapd* and *slurpd* communicate through a simple text file that is used to log changes.

See the Replication with slurpd chapter for information about how to configure and run *slurpd*(8).

---

2. A Quick-Start Guide

The following is a quick start guide to OpenLDAP 2.0 software, including the stand-alone LDAP daemon, *slapd*(8).

It is meant to step you through the basic steps needed to install and configure OpenLDAP software. It should be used in conjunction with the other chapters of this document, manual pages, and other materials provided with the distribution (e.g. the `INSTALL` document) or on the OpenLDAP web site (in particular, the OpenLDAP Software FAQ).

If you intend to run OpenLDAP seriously, you should review the all of this document before attempt to install the software.

Note: This quick start guide does not use strong authentication nor any privacy and integrity protection services. These services are described in other chapters of the OpenLDAP Administrator's Guide.

---

1. **Get the software**
   You can obtain a copy of the software by following the instructions on the OpenLDAP download page (http://www.openldap.org/software/download/). It is recommended that new users start with the (latest) *release*.

2. **Unpack the distribution**
   Pick a directory for the LDAP source to live under, change directory to there, and unpack the distribution using the following commands:
   ```
   gunzip -c openldap-VERSION.tgz | tar xvfB -
   ```
   then relocate yourself into the distribution directory:
   ```
   cd openldap-VERSION
   ```
   You'll have to replace `VERSION` with the version name of the release.

3. **Review documentation**
   You should now review the `COPYRIGHT`, `LICENSE`, `README` and `INSTALL` documents provided with the distribution. The `COPYRIGHT` and `LICENSE` provide information on acceptable use, copying, and limitation of warranty of OpenLDAP software.

   You should also review other chapters of this document. In particular, the Building and Installing OpenLDAP Software chapter of this document provides detailed information on prerequisite software and installation procedures.

4. **Run `configure`**
   You will need to run the provided `configure` script to *configure* the distribution for building on your system. The `configure` script accepts many command line options that enable or disable optional software features. Usually the defaults are okay, but you may want to change them. To get a complete list of options that `configure` accepts, use the `--help` option:
   ```
   ./configure --help
   ```
   However, given that you using this guide, we'll assume you'll are brave enough to just let `configure` to determine what's best:
   ```
   ./configure
   ```
   Assuming `configure` doesn't dislike your system, you can proceed with building the software. If `configure` did complain, well, you'll likely need to go to the FAQ Installation Section (http://www.openldap.org/faq/ and/or actually read the Building and Installing OpenLDAP Software chapter of this document.

5. **Build the software**.
   The next step is to build the software. This step has two parts, first we construct dependencies and then we compile the software:
   ```
   make depend
   make
   ```
   Both makes should complete without error.

6. **Test the build**.
   To ensure a correct build, you should run the test suite (it only takes a few minutes):
   ```
   make test
   ```
   Tests which apply to your configuration will run and they should pass. Some tests, such as the replication test, may be skipped.

7. **Install the software**.
   You are now ready to install the software, this usually requires *super-user* privledges:
   ```
   su root -c 'make install'
   ```
   Everything should now be installed under `/usr/local` (or whatever installation prefix was used by `configure`.

8. **Edit the configuration file**.
   Use your favorite editor to edit the provided *slapd.conf*(5) example (usually installed as `/usr/local/etc/openldap/slapd.conf`) to contain an LDBM database definition of the form:
   ```
   database ldbm
   suffix "dc=<MY-DOMAIN>,dc=<COM>"
   rootdn "cn=Manager,dc=<MY-DOMAIN>,dc=<COM>"
   rootpw secret
   directory /usr/local/var/openldap-ldbm
   ```
   Be sure to replace `<MY-DOMAIN>` and `<COM>` with the appropriate domain components of your domain name. For example, for `example.com`, use:
   ```
   database ldbm
   suffix "dc=example,dc=com"
   ```

```
rootdn "cn=Manager,dc=example,dc=com"
rootpw secret
directory /usr/local/var/openldap-ldbm
```

If your domain contains additional components, such as `eng.uni.edu.eu`, use:

```
database ldbm
suffix "dc=eng,dc=uni,dc=edu,dc=eu"
rootdn "cn=Manager,dc=eng,dc=uni,dc=edu,dc=eu"
rootpw secret
directory /usr/local/var/openldap-ldbm
```

Details regarding configuring *slapd*(8) can be found in the *slapd.conf*(5) manual page and the The slapd Configuration File chapter of this document.

9. **Start SLAPD**.
   You are now ready to start the stand-alone LDAP server, slapd(8), by running the command:
   ```
   su root -c /usr/local/libexec/slapd
   ```
   To check to see if the server is running and configured correctly, you can run a search against it with *ldapsearch*(1). By default, ldapsearch is installed as `/usr/local/bin/ldapsearch`:
   ```
   ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
   ```
   Note the use of single quotes around command parameters to prevent special characters from being interpreted by the shell. This should return:
   ```
   dn:
   namingContexts: dc=example,dc=com
   ```
   Details regarding running *slapd*(8) can be found in the *slapd*(8) manual page and the Running slapd chapter of this document.

10. **Add initial entries to your directory**.
    You can use *ldapadd*(1) to add entries to your LDAP directory. *ldapadd* expects input in LDIF form. We'll do it in two steps:
    1. create an LDIF file
    2. run ldapadd

    Use your favorite editor and create an LDIF file that contains:
    ```
    dn: dc=<MY-DOMAIN>,dc=<COM>
    objectclass: dcObject
    objectclass: organization
    o: <MY ORGANIZATION>
    dc: <MY-DOMAIN>

    dn: cn=Manager,dc=<MY-DOMAIN>,dc=<COM>
    objectclass: organizationalRole
    cn: Manager
    ```
    Be sure to replace <MY-DOMAIN> and <COM> with the appropriate domain components of your domain name. <MY ORGANIZATION> should be replaced with the name of your organization. If you cut and paste, be sure to trim any leading and trailing whitespace from the example.
    ```
    dn: dc=example,dc=com
    ```

```
objectclass: dcObject
objectclass: organization
o: Example Company
dc: example

dn: cn=Manager,dc=example,dc=com
objectclass: organizationalRole
cn: Manager
```

Now, you may run *ldapadd*(1) to insert these entries into your directory.

```
ldapadd -x -D "cn=Manager,dc=<MY-DOMAIN>,dc=<COM>" -W -f example.ldif
```

Be sure to replace `<MY-DOMAIN>` and `<COM>` with the appropriate domain components of your domain name. You will be prompted for the "`secret`" specified in `slapd.conf`. For example, for `example.com`, use:

```
ldapadd -x -D "cn=Manager,dc=example,dc=com" -W -f example.ldif
```

where `example.ldif` is the file you created above.

Additional informaton regarding directory creation can be found in the Database Creation and Maintenance Tools chapter of this document.

11. **See if it works**.
    Now we're ready to verify the added entries are in your directory. You can use any LDAP client to do this, but our example uses the *ldapsearch*(1) tool. Remember to replace `dc=example,dc=com` with the correct values for your site:

    ```
    ldapsearch -x -b 'dc=example,dc=com' '(objectclass=*)'
    ```

    This command will search for and retrieve every entry in the database.

You are now ready to add more entries using *ldapadd*(1) or another LDAP client, experiment with various configuration options, backend arrangements, etc.

Note that by default, the *slapd*(8) database grants *read access to everybody* excepting the *super-user* (as specified by the `rootdn` configuration directive). It is highly recommended that you establish controls to restrict access to authorized users. Access controls are discussed in the Access Control section of the The slapd Configuration File chapter.

The following chapters provide more detailed information on making, installing, and running *slapd*(8).

---

3. The Big Picture - Configuration Choices

This section gives a brief overview of various LDAP directory configurations, and how your stand-alone LDAP server *slapd*(8) fits in with the rest of the world.

3.1. Local Directory Service

In this configuration, you run a *slapd* which provides directory service for your local domain only. It does not interact with other directory servers in any way. This configuration is shown in Figure 3.1.

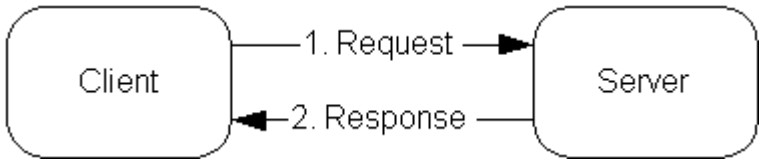

Figure 3.1: Local service configuration.

Use this configuration if you are just starting out (it's the one the quick-start guide makes for you) or if you want to provide a local service and are not interested in connecting to the rest of the world. It's easy to upgrade to another configuration later if you want.

3.2. Local Directory Service with Referrals

In this configuration, you run a slapd which provides directory service for your local domain and configure it to return referrals to a *superior* service capable of requests outside your local domain. You may run this service yourself or use one provided to you. This configuration is shown in Figure 3.2.

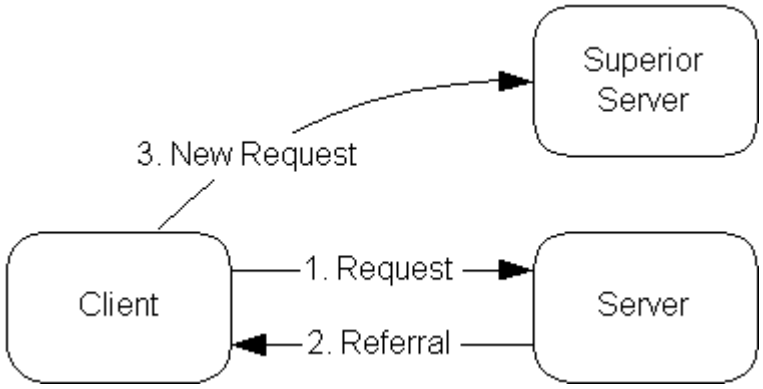

Figure 3.2: Local service with referrals

Use this configuration if you want to provide local service and participate in the Global Directory.

3.3. Replicated Directory Service

The slurpd daemon is used to propagate changes from a master slapd to one or more slave slapds. An example master-slave configuration is shown in figure 3.3.

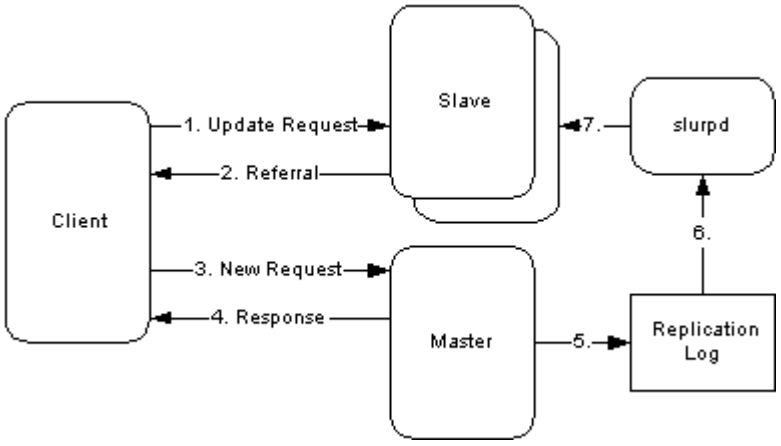

Figure 3.3: Replicated Directory Services

This configuration can be used in conjunction with either of first two configurations in situations where a single slapd does not provide the required reliability or availability.

3.4. Distributed Local Directory Service

In this configuration, the local service is partitioned into smaller services, each which may be replicated, and *glued* together with *superior* and *subordinate* referrals.

---

4. Building and Installing OpenLDAP Software

This chapter details how to build and install the OpenLDAP Software package including *slapd*(8), the stand-alone LDAP daemon and *slurpd*(8), the stand-alone update replication daemon. Building and installing OpenLDAP requires several steps: installing prerequisite software, configuring OpenLDAP itself, making, and finally installing. The following sections describe this process in detail.

4.1. Obtaining and Extracting the Software

You can obtain OpenLDAP Software from the project's download page at http://www.openldap.org/software/download/ or directly from the project's FTP service at ftp://ftp.openldap.org/pub/OpenLDAP/.

The project makes available two series of packages for *general use*. The project makes *releases* as new features and bug fixes come available. Though the project takes steps to improve stablity of these releases, it is common for problems to arise only after *release*. The latest *release* which has demonstrated stability through general use.

Users of OpenLDAP Software can choose, depending on their desire for the *latest features* versus *demonstrated stability*, the most appropriate series to install.

After downloading OpenLDAP Software, you need to extract the distribution from the compressed archive file and change your working directory to the top directory of the distribution:

```
gunzip -c openldap-VERSION.tgz | tar xf -
cd openldap-VERSION
```

You'll have to replace `VERSION` with the version name of the release.

You should now review the `COPYRIGHT`, `LICENSE`, `README` and `INSTALL` documents provided with the distribution. The `COPYRIGHT` and `LICENSE` provide information on acceptable use, copying, and limitation of warranty of OpenLDAP software. The `README` and `INSTALL` documents provide detailed information on prerequisite software and installation procedures.

4.2. Prerequisite software

OpenLDAP Software relies upon a number of software packages distributed by third parties. Depending on the features you intend to use, you may have to download and install a number of additional software packages. This section details commonly needed third party software packages you might have to install. Note that some of these third party packages may depend on additional software packages. Install each package per installation instructions provided with it.

4.2.1. Transport Layer Security

OpenLDAP clients and servers require installation of OpenSSL TLS libraries to provide Transport Layer Security services. Though some operating systems may provide these libraries as part of the base system or as an optional software component, OpenSSL often requires separate installation.

OpenSSL is available from http://www.openssl.org/.

OpenLDAP will not be fully LDAPv3 compliant unless OpenLDAP's `configure` detects a usable OpenSSL installation.

4.2.2. Kerberos Authentication Services

OpenLDAP clients and servers support Kerberos-based authentication services. In particular, OpenLDAP supports SASL/GSSAPI authentication mechanism using either Heimdal or MIT Kerberos V packages. If you desire to use Kerberos-based SASL/GSSAPI authentication, you should install either Heimdal or MIT Kerberos V.

Heimdal Kerberos is available from http://www.pdc.kth.se/heimdal/. MIT Kerberos is available from http://web.mit.edu/kerberos/www/.

Use of strong authentication services, such as those provided by Kerberos, is highly recommended.

4.2.3. Simple Authentication and Security Layer

OpenLDAP clients and servers require installation of Cyrus's SASL libraries to provide Simple Authentication and Security Layer services. Though some operating systems may provide this library as part of the base system or as an optional software component, Cyrus SASL often requires separate installation.

Cyrus SASL is available from http://asg.web.cmu.edu/sasl/sasl-library.html. Cyrus SASL will make use of OpenSSL and Kerberos/GSSAPI libraries if preinstalled.

OpenLDAP will not be fully LDAPv3 compliant unless OpenLDAP's configure detects a usable Cyrus SASL installation.

4.2.4. Database Software

OpenLDAP's *slapd*(8) primary database backend, LDBM, requires a compatible database package for entry storage. LDBM is compatible with Sleepycat Software's BerkeleyDB (recommended) or the Free Software Foundation's GNU Database Manager (GDBM). If neither of these packages are available at configure time, you will not be able build *slapd*(8) with primary database backend.

Your operating system may provide one of these two packages in the base system or as an optional software component. You may need may need to obtain the software and install it yourself.

BerkeleyDB is available from Sleepycat Software's download page http://www.sleepycat.com/download.html. There are several versions available. At the time of this writing, the latest release, version 3.1, is recommended.

GDBM is available from FSF's download site ftp://ftp.gnu.org/pub/gnu/gdbm/. At the time of this writing, version 1.8 is the latest release.

4.2.5. Threads

OpenLDAP is designed to take advantage of threads. OpenLDAP supports POSIX *pthreads*, Mach *CThreads*, and a number of other varieties. `configure` will complain if it cannot find a suitable thread subsystem. If this occurs, please consult the `Software|Installation|Platform Hints` section of the OpenLDAP FAQ http://www.openldap.org/faq/.

4.2.6. TCP Wrappers

slapd(8) supports TCP wrappers (IP level access control filters) if preinstalled. Use of TCP wrappers or other IP-level access filters (such as those provided by an IP-level firewall) is recommended for servers containing non-public information.

4.3. Running configure

Now you should probably run the `configure` script with the `--help` option. This will give you a list of options that you can change when building OpenLDAP. Many of the features of OpenLDAP can be enabled or disabled using this method.

```
./configure --help
```

The `configure` script will also look at various environment variables for certain settings. These environment variables include:

Table 4.1: Environment Variables

Variable	Description
CC	Specify alternative C Compiler
CFLAGS	Specify additional compiler flags
CPPFLAGS	Specify C Preprocessor flags
LDFLAGS	Specify linker flags
LIBS	Specify additional libraries

Now run the configure script with any desired configuration options or environment variables.

```
[[env] settings] ./configure [options]
```

As an example, let's assume that we want install OpenLDAP with LDBM backend and TCP wrapper support. By default, LDBM is enabled and TCP wrappers is not. So, we just need to specify `--with-wrappers` to include TCP wrapper support:

```
./configure --with-wrappers
```

However, this will fail to locate dependent software not installed in system directories. For example, if TCP Wrappers headers and libraries are installed in `/usr/local/include` and `/usr/local/lib` respectively, the `configure` script should be called as follows:

```
env CPPFLAGS="-I/usr/local/include" LDFLAGS="-L/usr/local/lib" \
        ./configure --with-wrappers
```

Note: Some shells, such as those derived from the Bourne *sh*(1), do not require use of the *env*(1) command. In some cases, environmental variables have to be specified using alternative syntaxes.

The `configure` script will normally auto-detect appropriate settings. If you have problems at this stage, consult any platform specific hints and check your `configure` options, if any.

4.4. Building the Software

Once you have run the `configure` script the last line of output should be:

```
Please "make depend" to build dependencies
```

If the last line of output does not match, `configure` has failed, and you will need to review its output to determine what went wrong. You should not proceed until `configure` completes successfully.

To build dependencies, run:

```
make depend
```

Now build the software, this step will actually compile OpenLDAP.

```
make
```

You should examine the output of this command carefully to make sure everything is built correctly. Note that this command builds the LDAP libraries and associated clients as well as *slapd*(8) and *slurpd*(8).

4.5. Testing the Software

Once the software has been properly configured and successfully made, you should run the test suite to verify the build.

```
make test
```

Tests which apply to your configuration will run and they should pass. Some tests, such as the replication test, may be skipped if not supported by your configuration.

4.6. Installing the Software

One you have successfully tested the software, you are ready to install it. You will need to have write permission to the installation directories you specified when you ran configure. By default OpenLDAP is installed in `/usr/local`. If you changed this setting with the `--prefix` configure option, it will be installed in the location you provided.

Typically, the installation typically requires super-user priviledges. From the top level OpenLDAP source directory, type:

```
su root -c 'make install'
```

You should examine the output of this command carefully to make sure everything is installed correctly. You will find the configuration files for *slapd*(8) in `/usr/local/etc/openldap` by default. See the The slapd Configuration File chapter for additional information.

---

5. The slapd Configuration File

Once the software has been built and installed, you are ready to configure *slapd*(8) for use at your site. The slapd runtime configuration is primarily accomplished through the *slapd.conf*(5) file, normally installed in the `/usr/local/etc/openldap` directory.

An alternate configuration file can be specified via a command-line option to *slapd*(8) or *slurpd*(8). This chapter describes the general format of the config file, followed by a detailed description of commonly used config file directives.

5.1. Configuration File Format

The *slapd.conf*(5) file consists of three types of configuration information: global, backend specific, and database specific. Global information is specified first, followed by information associated with a particular backend type, which is then followed by information associated with a particular database instance. Global directives can be overridden in a backend and/or database directives, backend directives can be overridden by database directives.

Blank lines and comment lines beginning with a '#' character are ignored. If a line begins with white space, it is considered a continuation of the previous line. The general format of slapd.conf is as follows:

```
# global configuration directives
<global config directives>

# backend definition
backend <typeA>
<backend-specific directives>

# first database definition & config directives
database <typeA>
<database-specific directives>

# second database definition & config directives
database <typeB>
<database-specific directives>

# second database definition & config directives
database <typeA>
<database-specific directives>

# subsequent backend & database definitions & config directives
...
```

A configuration directive may take arguments. If so, they are separated by white space. If an argument contains white space, the argument should be enclosed in double quotes `"like this"`. If an argument contains a double quote or a backslash character `\', the character should be preceded by a backslash character `\'.

The distribution contains an example configuration file that will be installed in the `/usr/local/etc/openldap` directory. A number of files containing schema definitions (attribute types and object classes) are also provided in the `/usr/local/etc/openldap/schema` directory.

5.2. Configuration File Directives

This section details commonly used configuration directives. For a complete list, see *slapd.conf*(5) manual page. This section separates the configuration file directives into global, backend-specific and data-specific categories, describing each directive and its default value (if any), and giving an example of its use.

5.2.1. Global Directives

Directives described in this section apply to all backends and databases unless specifically overridden in a backend or database definition. Arguments that should be replaced by actual text are shown in brackets <>.

5.2.1.1. access to <what> [by <who> <accesslevel> <control>]+

This directive grants access (specified by <accesslevel>) to a set of entries and/or attributes (specified by <what>) by one or more requesters (specified by <who>). See the Access Control section of this chapter for a summary of basic usage.

5.2.1.2. attributetype <RFC2252 Attribute Type Description>

This directive defines an attribute type. Please see the Schema Specification chapter for information regarding how to use this directive.

5.2.1.3. defaultaccess { none | compare | search | read | write }

This directive specifies the default access to grant requesters when no `access` directives have been specified. Any given access level implies all lesser access levels (e.g., read access implies search and compare but not write).

Note: It is recommend that the `access` directive be used to specify access control. See the Access Control section of this chapter for information regarding the `access` directive.

Default:

```
defaultaccess read
```

5.2.1.4. idletimeout <integer>

Specify the number of seconds to wait before forcibly closing an idle client connection. An idletimeout of 0, the default, disables this feature.

5.2.1.5. include <filename>

This directive specifies that slapd should read additional configuration information from the given file before continuing with the next line of the current file. The included file should follow the normal slapd config file format. The file is commonly used to include files containing schema specifications.

Note: You should be careful when using this directive - there is no small limit on the number of nested include directives, and no loop detection is done.

5.2.1.6. loglevel <integer>

This directive specifies the level at which debugging statements and operation statistics should be syslogged (currently logged to the *syslogd*(8) {EX:LOG_LOCAL4}} facility). You must have configured OpenLDAP `--enable-debug` (the default) for this to work (except for the two statistics levels, which are always enabled). Log levels are additive. To display what numbers correspond to what kind of debugging, invoke slapd with `-?` or consult the table below. The possible values for <integer> are:

Table 5.1: Debugging Levels

Level	Description
-1	enable all debugging
0	no debugging
1	trace function calls
2	debug packet handling
4	heavy trace debugging
8	connection management
16	print out packets sent and received
32	search filter processing
64	configuration file processing
128	access control list processing
256	stats log connections/operations/results
512	stats log entries sent
1024	print communication with shell backends
2048	print entry parsing debugging

Example:

```
loglevel -1
```

This will cause lots and lots of debugging information to be logged.

Default:

```
loglevel 256
```

5.2.1.7. objectclass <RFC2252 Object Class Description>

This directive defines an object class. Please see the Schema Specification chapter for information regarding how to use this directive.

5.2.1.8. referral <URI>

This directive specifies the referral to pass back when slapd cannot find a local database to handle a request.

Example:

```
        referral ldap://root.openldap.org
```

This will refer non-local queries to the global root LDAP server at the OpenLDAP Project. Smart LDAP clients can re-ask their query at that server, but note that most of these clients are only going to know how to handle simple LDAP URLs that contain a host part and optionally a distinguished name part.

5.2.1.9. sizelimit <integer>

This directive specifies the maximum number of entries to return from a search operation.

Default:

```
sizelimit 500
```

5.2.1.10. timelimit <integer>

This directive specifies the maximum number of seconds (in real time) slapd will spend answering a search request. If a request is not finished in this time, a result indicating an exceeded timelimit will be returned.

Default:

```
timelimit 3600
```

5.2.2. General Backend Directives

Directives in this section apply only to the backend in which they are defined. They are supported by every type of backend. Backend directives apply to all databases instances of the same type and, depending on the directive, may be overridden by database directives.

5.2.2.1. backend <type>

This directive marks the beginning of a backend definition. `<type>` should be one of `ldbm`, `shell`, `passwd`, or other supported backend type.

5.2.3. General Database Directives

Directives in this section apply only to the database in which they are defined. They are supported by every type of database.

5.2.3.1. database <type>

This directive marks the beginning of a new database instance definition. `<type>` should be one of `ldbm`, `shell`, `passwd`, or other supported database type.

Example:

```
database ldbm
```

This marks the beginning of a new LDBM backend database instance definition.

5.2.3.2. readonly { on | off }

This directive puts the database into "read-only" mode. Any attempts to modify the database will return an "unwilling to perform" error.

Default:

```
readonly off
```

5.2.3.3. replica

```
replica host=<hostname>[:<port>]
        [bindmethod={ simple | kerberos | sasl }]
        ["binddn=<DN>"]
        [mech=<mech>]
        [authcid=<identity>]
        [authzid=<identity>]
        [credentials=<password>]
        [srvtab=<filename>]
```

This directive specifies a replication site for this database. The `host=` parameter specifies a host and optionally a port where the slave slapd instance can be found. Either a domain name or IP address may be used for <hostname>. If <port> is not given, the standard LDAP port number (389) is used.

The `binddn=` parameter gives the DN to bind as for updates to the slave slapd. It should be a DN which has read/write access to the slave slapd's database, typically given as a `rootdn` in the slave's config file. It must also match the `updatedn` directive in the slave slapd's config file. Since DNs are likely to contain embedded spaces, the entire `"binddn=<DN>"` string should be enclosed in double quotes.

The `bindmethod` is `simple` or `kerberos` or `sasl`, depending on whether simple password-based authentication or Kerberos authentication or SASL authentication is to be used when connecting to the slave slapd.

Simple authentication should not be used unless adequate integrity and privacy protections are in place (e.g. TLS or IPSEC). Simple authentication requires specification of `binddn` and `credentials` parameters.

Kerberos authentication is deprecated in favor of SASL authentication mechanisms, in particular the `KERBEROS_V4` and `GSSAPI` mechanisms. Kerberos authentication requires `binddn` and `srvtab` parameters.

SASL authentication is generally recommended. SASL authentication requires specification of a mechanism using the `mech` parameter. Depending on the mechanism, an authentication identity and/or credentials can be specified using `authcid` and `credentials` respectively. The `authzid` parameter may be used to specify an authorization identity.

See the chapter entitled Replication with slurpd for more information on how to use this directive.

5.2.3.4. replogfile <filename>

This directive specifies the name of the replication log file to which slapd will log changes. The replication log is typically written by slapd and read by slurpd. Normally, this directive is only used if slurpd is being used to replicate the database. However, you can also use it to generate a transaction log, if slurpd is not running. In this case, you will need to periodically truncate the file, since it will grow indefinitely otherwise.

See the chapter entitled Replication with slurpd for more information on how to use this directive.

5.2.3.5. rootdn <dn>

This directive specifies the DN that is not subject to access control or administrative limit restrictions for operations on this database. The DN need not refer to an entry in the directory. The DN may refer to a SASL identity.

Entry-based Example:

```
rootdn "cn=Manager,dc=example,dc=com"
```

SASL-based Example:

```
rootdn "uid=root@EXAMPLE.COM"
```

5.2.3.6. rootpw <password>

This directive specifies a password for the DN given above that will always work, regardless of whether an entry with the given DN exists or has a password. This directive is deprecated in favor of SASL based authentication.

Example:

```
rootpw secret
```

5.2.3.7. suffix <dn suffix>

This directive specifies the DN suffix of queries that will be passed to this backend database. Multiple suffix lines can be given, and at least one is required for each database definition.

Example:

```
suffix "dc=example,dc=com"
```

Queries with a DN ending in "dc=example,dc=com" will be passed to this backend.

Note: When the backend to pass a query to is selected, slapd looks at the suffix line(s) in each database definition in the order they appear in the file. Thus, if one database suffix is a prefix of another, it must appear after it in the config file.

5.2.3.8. updatedn <dn>

This directive is only applicable in a slave slapd. It specifies the DN allowed to make changes to the replica. This may be the DN *slurpd*(8) binds as when making changes to the replica or the DN associated with a SASL identity.

Entry-based Example:

```
updatedn "cn=Update Daemon,dc=example,dc=com"
```

SASL-based Example:

```
updatedn "uid=slurpd@EXAMPLE.COM"
```

See the Replication chapter for more information on how to use this directive.

5.2.3.9. updateref <URL>

This directive is only applicable in a slave slapd. It specifies the URL to return to clients which submit update requests upon the replica. If specified multiple times, each URL is provided.

Example:

```
updateref       ldap://master.example.net
```

5.2.4. LDBM Backend-Specific Directives

Directives in this category only apply to the LDBM backend database. That is, they must follow a "database ldbm" line and come before any other "database" line.

5.2.4.1. cachesize <integer>

This directive specifies the size in entries of the in-memory cache maintained by the LDBM backend database instance.

Default:

```
cachesize 1000
```

5.2.4.2. dbcachesize <integer>

This directive specifies the size in bytes of the in-memory cache associated with each open index file. If not supported by the underlying database method, this directive is ignored without comment. Increasing this number uses more memory but can cause a dramatic performance increase, especially during modifies or when building indexes.

Default:

```
dbcachesize 100000
```

5.2.4.3. dbnolocking

This option, if present, disables database locking. Enabling this option may improve performance at the expense of data security.

5.2.4.4. dbnosync

This option causes on-disk database contents not be immediately synchronized with in memory changes upon change. Enabling this option may improve performance at the expense of data security.

5.2.4.5. directory <directory>

This directive specifies the directory where the LDBM files containing the database and associated indexes live.

Default:

```
directory /usr/local/var/openldap-ldbm
```

5.2.4.6. index {<attrlist> | default} [pres,eq,approx,sub,none]

This directive specifies the indexes to maintain for the given attribute. If only an `<attrlist>` is given, the default indexes are maintained.

Example:

```
index default pres,eq
index objectClass,uid
index cn,sn eq,sub,approx
```

The first line sets the default set of indices to maintain to present and equality. The second line causes the default (pres,eq) set of indices to be maintained for `objectClass` and `uid` attribute types. The third line causes equality, substring, and approximate indices to be maintained for `cn` and `sn` attribute types.

5.2.4.7. mode <integer>

This directive specifies the file protection mode that newly created database index files should have.

Default:

```
mode 0600
```

5.2.5. Other Backend Databases

slapd(8) supports a number of backend database types besides the default LDBM.

Table 5.2: Backend Database Types

Types	Description
`ldbm`	Berkeley or GNU DBM compatible backend
`passwd`	Provides read-only access to `/etc/passwd`
`shell`	Shell (extern program) backend
`sql`	SQL Programmable backend

See *slapd.conf*(5) for details.

5.3. Access Control

Access to slapd entries and attributes is controlled by the access configuration file directive. The general form of an access line is:

```
<access directive> ::= access to <what>
        [by <who> <access> <control>]+
<what> ::= * | [ dn[.<target style>]=<regex>]
```

```
        [filter=<ldapfilter>] [attrs=<attrlist>]
<target style> ::= regex | base | one | subtree | children
<attrlist> ::= <attr> | <attr> , <attrlist>
<attr> ::= <attrname> | entry | children
<who> ::= [* | anonymous | users | self |
        dn[.<subject style>]=<regex>]
        [dnattr=<attrname> ]
        [group[/<objectclass>[/<attrname>][.<basic style>]]=<regex> ]
        [peername[.<basic style>]=<regex>]
        [sockname[.<basic style>]=<regex>]
        [domain[.<basic style>]=<regex>]
        [sockurl[.<basic style>]=<regex>]
        [set=<setspec>]
        [aci=<attrname>]
<subject style> ::= regex | exact | base | one | subtree | children
<basic style> ::= regex | exact
<access> ::= [self]{<level>|<priv>}
<level> ::= none | auth | compare | search | read | write
<priv> ::= {=|+|-}{w|r|s|c|x}+
<control> ::= [stop | continue | break]
```

where the <what> part selects the entries and/or attributes to which the access applies, the `<who>` part specifies which entities are granted access, and the `<access>` part specifies the access granted. Multiple `<who> <access> <control>` triplets are supported, allowing many entities to be granted different access to the same set of entries and attributes.

5.3.1. What to control access to

The <what> part of an access specification determines the entries and attributes to which the access control applies. Entries can be selected in two ways: by a regular expression matching the entry's distinguished name:

```
dn=<regular expression>
```

Note: The DN pattern specified should be "normalized" to the RFC2253 restricted DN form. In particular, there should be no extra spaces and commas should be used to separate components. An example normalized DN is "`cn=Babs Jensen,dc=example,dc=com`". An example of a non-normalized DN is "`cn=Babs Jensen; dc=example; dc=com`".

Or, entries may be selected by a filter matching some attribute(s) in the entry:

```
filter=<ldap filter>
```

where <ldap filter> is a string representation of an LDAP search filter, as described in RFC2254.

Attributes within an entry are selected by including a comma-separated list of attribute names in the <what> selector:

```
attrs=<attribute list>
```

Access to the entry itself must be granted or denied using the special attribute name "`entry`". Note that giving access to an attribute is not enough; access to the entry itself through the `entry` attribute is also required. The complete examples at the end of this section should help clear things up.

Lastly, there is a special entry selector "*" that is used to select any entry. It is used when no other <what> selector has been provided. It's equivalent to "dn=.*"

5.3.2. Who to grant access to

The <who> part identifies the entity or entities being granted access. Note that access is granted to "entities" not "entries." The following table summarizes entity specifiers:

Table 5.3: Access Entity Specifiers

Specifier	Entities
*	All, including anonymous and authenticated users
anonymous	Anonymous (non-authenticated) users
users	Authenticated users
self	User associated with target entry
dn=<regex>	Users matching regular expression

The DN specifier takes a regular expression which is used to match against the "normalized" DN of the current entity.

```
dn=<regular expression>
```

By "normalized", we mean that all extra spaces have been removed from the entity's DN and commas are used to separate RDN components.

Other control factors are also supported. For example, a <what> can be restricted by a regular expression matching the client's domain name:

```
domain=<regular expression>
```

or by an entry listed in a DN-valued attribute in the entry to which the access applies:

```
dnattr=<dn-valued attribute name>
```

The dnattr specification is used to give access to an entry whose DN is listed in an attribute of the entry (e.g., give access to a group entry to whoever is listed as the owner of the group entry).

5.3.3. The access to grant

The kind of <access> granted can be one of the following:

Table 5.4: Access Levels

Level	Privileges	Description
none		no access
auth	=x	needed to bind
compare	=cx	needed to compare
search	=scx	needed to apply search filters
read	=rscx	needed to read search results
write	=wrscx	needed to modify/rename

Each level implies all lower levels of access. So, for example, granting someone write access to an entry also grants them read, search, compare, and auth access. However, one may use the privileges specifier to grant specific permissions.

5.3.4. Access Control Evaluation

When evaluating whether some requester should be given access to an entry and/or attribute, slapd compares the entry and/or attribute to the <what> selectors given in the configuration file. Access directives local to the current database are examined first, followed by global access directives. Within this priority, access directives are examined in the order in which they appear in the config file. Slapd stops with the first <what> selector that matches the entry and/or attribute. The corresponding access directive is the one slapd will use to evaluate access.

Next, slapd compares the entity requesting access to the <who> selectors within the access directive selected above in the order in which they appear. It stops with the first <who> selector that matches the requester. This determines the access the entity requesting access has to the entry and/or attribute.

Finally, slapd compares the access granted in the selected <access> clause to the access requested by the client. If it allows greater or equal access, access is granted. Otherwise, access is denied.

The order of evaluation of access directives makes their placement in the configuration file important. If one access directive is more specific than another in terms of the entries it selects, it should appear first in the config file. Similarly, if one <who> selector is more specific than another it should come first in the access directive. The access control examples given below should help make this clear.

5.3.5. Access Control Examples

The access control facility described above is quite powerful. This section shows some examples of its use. First, some simple examples:

```
access to * by * read
```

This access directive grants read access to everyone.

```
access to *
        by self write
        by anonymous auth
        by * read
```

This directive allows users to modify their own entries, allows authenticate, and allows authenticated users to read. Note that only the first `by <who>` clause which matches applies. Hence, the anonymous users are granted `auth`, not `read`. The last clause could just as well have been "`by users read`".

The following example shows the use of a regular expression to select the entries by DN in two access directives where ordering is significant.

```
access to dn=".*,dc=example,dc=com"
        by * search
access to dn=".*,dc=com"
        by * read
```

Read access is granted to entries under the `dc=com` subtree, except for those entries under the `dc=example,dc=com` subtree, to which search access is granted. No access is granted to `dc=com` as neither access directive matches this DN. If the order of these access directives was reversed, the trailing directive would never be reached, since all `dc=example,dc=com` entries are also `dc=com` entries.

Also note that if no `access to` directive matches or no `by <who>` clause, **access is denied**. That is, every `access to` directive ends with an implicit `by * none` clause and every access list ends with an implicit `access to * by * none` directive. Only if no access controls are specified is the `defaultaccess` granted.

The next example again shows the importance of ordering, both of the access directives and the `by <who>` clauses. It also shows the use of an attribute selector to grant access to a specific attribute and various `<who>` selectors.

```
access to dn="(.*,)?dc=example,dc=com" attr=homePhone
        by self write
        by dn="(.*,)?dc=example,dc=com" search
        by domain=.*\.example\.com read
access to dn="(.*,)?dc=example,dc=com"
        by self write
        by dn=".*,dc=example,dc=com" search
        by anonymous auth
```

This example applies to entries in the "`dc=example,dc=com`" subtree. To all attributes except `homePhone`, the entry itself can write them, other `example.com` entries can search by them, anybody else has no access ((implicit `by * none`) excepting for authentication/authorization (which is always done anonymously). The `homePhone` attribute is writable by the entry, searchable by other `example.com` entries, readable by clients connecting from somewhere in the `example.com` domain, and otherwise not readable (implicit `by * none`). All other access is denied by the implicit `access to * by * none`.

Sometimes it is useful to permit a particular DN to add or remove itself from an attribute. For example, if you would like to create a group and allow people to add and remove only their own DN from the member attribute, you could accomplish it with an access directive like this:

```
access to attr=member,entry
        by dnattr=member selfwrite
```

The dnattr `<who>` selector says that the access applies to entries listed in the `member` attribute. The `selfwrite` access selector says that such members can only add or delete their own DN from the attribute, not other values. The addition of the entry attribute is required because access to the entry is required to access any of the entry's attributes.

5.4. Configuration File Example

The following is an example configuration file, interspersed with explanatory text. It defines two databases to handle different parts of the X.500 tree; both are LDBM database instances. The line numbers shown are provided for reference only and are not included in the actual file. First, the global configuration section:

```
1.    # example config file - global configuration section
2.    include /usr/local/etc/schema/core.schema
3.    referral ldap://root.openldap.org
4.    access to * by * read
```

Line 1 is a comment. Line 2 includes another config file which containing *core* schema definitions. The `referral` directive on line 3 means that queries not local to one of the databases defined below will be referred to the LDAP server running on the standard port (389) at the host `root.openldap.org`.

Line 4 is a global access control. It is used only if no database access controls match or when the target objects are not under the control of any database (such as the Root DSE).

The next section of the configuration file defines an LDBM backend that will handle queries for things in the "dc=example,dc=com" portion of the tree. The database is to be replicated to two slave slapds, one on truelies, the other on judgmentday. Indexes are to be maintained for several attributes, and the `userPassword` attribute is to be protected from unauthorized access.

```
5.    # ldbm definition for the example.com
6.    database ldbm
7.    suffix "dc=example,dc=com"
8.    directory /usr/local/var/openldap
9.    rootdn "cn=Manager,dc=example,dc=com"
10.   rootpw secret
11.   # replication directives
12.   replogfile /usr/local/var/openldap/slapd.replog
13.   replica host=slave1.example.com:389
14.           binddn="cn=Replicator,dc=example,dc=com"
15.           bindmethod=simple credentials=secret
16.   replica host=slave2.example.com
17.           binddn="cn=Replicator,dc=example,dc=com"
18.           bindmethod=simple credentials=secret
19.   # indexed attribute definitions
20.   index uid pres,eq
21.   index cn,sn,uid pres,eq,approx,sub
22.   index objectClass eq
23.   # ldbm access control definitions
24.   access to attr=userPassword
25.           by self write
26.           by anonymous auth
27.           by dn="cn=Admin,dc=example,dc=com" write
28.           by * none
29.   access to *
30.           by self write
31.           by dn="cn=Admin,dc=example,dc=com" write
32.           by users read
```

Line 5 is a comment. The start of the database definition is marked by the database keyword on line 6. Line 7 specifies the DN suffix for queries to pass to this database. Line 8 specifies the directory in which the database files will live.

Lines 9 and 10 identify the database "super user" entry and associated password. This entry is not subject to access control or size or time limit restrictions.

Lines 11 through 18 are for replication. Line 11 specifies the replication log file (where changes to the database are logged - this file is written by slapd and read by slurpd). Lines 12 through 14 specify the hostname and port for a replicated host, the DN to bind as when performing updates, the bind method (simple) and the credentials (password) for the binddn. Lines 15 through 18 specify a second replication site. See the Replication with slurpd chapter for more information on these directives.

Lines 20 through 22 indicate the indexes to maintain for various attributes.

Lines 24 through 32 specify access control for entries in the database. For all entries, the `userPassword` attribute is writable by the entry itself and by the "admin" entry. It may be used for authentication/authorization purposes, but is otherwise not readable. All other attributes are writable by the entry and the "admin" entry, but may be read by authenticated users.

The next section of the example configuration file defines another LDBM database. This one handles queries involving the `dc=example,dc=net` subtree. Note that without line 38, the read access would be allowed due to the global access rule at line 4.

```
33.    # ldbm definition for example.net
34.    database ldbm
35.    suffix "dc=example,dc=net"
36.    directory /usr/local/var/ldbm-example-net
37.    rootdn "cn=Manager,dc=example,dc=com"
38.    access to * by users read
```

---

6. Running slapd

slapd(8) is designed to be run as a stand-alone server. This allows the server to take advantage of caching, manage concurrency issues with underlying databases, and conserve system resources. Running from *inetd*(8) is *NOT* an option.

6.1. Command-Line Options

slapd(8) supports a number of command-line options as detailed in the manual page. This section details a few commonly used options.

```
-f <filename>
```

This option specifies an alternate configuration file for slapd. The default is normally `/usr/local/etc/openldap/slapd.conf`.

```
-h <URLs>
```

This option specifies alternative listener configurations. The default is `ldap:///` which implies LDAP over TCP on all interfaces on the default LDAP port 389. You can specify specific host-port pairs or other protocol schemes (such as ldaps:// or ldapi://). For example, `-h "ldaps://`

`ldap://127.0.0.1:666"` will create two listeners: one for LDAP over SSL on all interfaces on the default LDAP/SSL port 636, and one for LDAP over TCP on the `localhost` (*loopback*) interface on port 666. Hosts may be specified using IPv4 dotted-decimal form or using host names. Port values must be numeric.

```
-n <service-name>
```

This option specifies the service name used for logging and other purposes. The default service name is `slapd`.

```
-l <syslog-local-user>
```

This option specifies the local user for the *syslog*(8) facility. Values can be `LOCAL0`, `LOCAL1`, `LOCAL2`, ..., and `LOCAL7`. The default is `LOCAL4`. This option may not be supported on all systems.

```
-u user -g group
```

These options specify the user and group, respectively, to run as. `user` can be either a user name or uid. `group` can be either a group name or gid.

```
-r directory
```

This option specifies a run-time directory. slapd will *chroot*(2) to this directory after opening listeners but before reading any configuration files or initializing any backends.

```
-d <level> | ?
```

This option sets the slapd debug level to <level>. When level is a `?' character, the various debugging levels are printed and slapd exits, regardless of any other options you give it. Current debugging levels are

Table 6.1: Debugging Levels

Level	Description
-1	enable all debugging
0	no debugging
1	trace function calls
2	debug packet handling
4	heavy trace debugging
8	connection management
16	print out packets sent and received
32	search filter processing
64	configuration file processing
128	access control list processing
256	stats log connections/operations/results
512	stats log entries sent
1024	print communication with shell backends
2048	print entry parsing debugging

You may enable multiple levels by specifying the debug option once for each desired level. Or, since debugging levels are additive, you can do the math yourself. That is, if you want to trace function calls and watch the config file being processed, you could set level to the sum of those two levels (in this case, `-d 65`). Or, you can let slapd do the math, (e.g. `-d 1 -d 64`). Consult `<ldap.h>` for more details.

Note: slapd must have been compiled with `-DLDAP_DEBUG` defined for any debugging information beyond the two stats levels to be available.

6.2. Starting slapd

In general, slapd is run like this:

```
/usr/local/etc/libexec/slapd [<option>]*
```

where `/usr/local/etc/libexec` is determined by `configure` and <option> is one of the options described above (or in *slapd*(8)). Unless you have specified a debugging level (including level `0`), slapd will automatically fork and detach itself from its controlling terminal and run in the background.

6.3. Stopping slapd

To kill off slapd safely, you should give a command like this

```
kill -INT `cat /usr/local/var/slapd.pid`
```

where `/usr/local/var` is determined by `configure`.

Killing slapd by a more drastic method may cause information loss or database corruption.

---

7. Database Creation and Maintenance Tools

This section tells you how to create a slapd database from scratch, and how to do trouble shooting if you run into problems. There are two ways to create a database. First, you can create the database on-line using LDAP. With this method, you simply start up slapd and add entries using the LDAP client of your choice. This method is fine for relatively small databases (a few hundred or thousand entries, depending on your requirements). This method works for database types which support updates.

The second method of database creation is to do it off-line using special utilities provided with slapd. This method is best if you have many thousands of entries to create, which would take an unacceptably long time using the LDAP method, or if you want to ensure the database is not accessed while it is being created. Note that not all database types support these utilitites.

7.1. Creating a database over LDAP

With this method, you use the LDAP client of your choice (e.g., the *ldapadd*(1)) to add entries, just like you would once the database is created. You should be sure to set the following options in the configuration file before starting *slapd*(8).

```
suffix <dn>
```

As described in the General Database Directives section, this option defines which entries are to be held by this database. You should set this to the DN of the root of the subtree you are trying to create. For example:

```
suffix "dc=example,dc=com"
```

You should be sure to specify a directory where the index files should be created:

```
directory <directory>
```

For example:

```
directory /usr/local/var/openldap-ldbm
```

You need to create this directory with appropriate permissions such that slapd can write to it.

You need to configure slapd so that you can connect to it as a directory user with permission to add entries. You can configure the directory to support a special *super-user* or *root* user just for this purpose. This is done through the following two options in the database definition:

```
rootdn <dn>
rootpw <passwd>
```

For example:

```
rootdn "cn=Manager,dc=example,dc=com"
rootpw secret
```

These options specify a DN and password that can be used to authenticate as the *super-user* entry of the database (i.e., the entry allowed to do anything). The DN and password specified here will always work, regardless of whether the entry named actually exists or has the password given. This solves the chicken-and-egg problem of how to authenticate and add entries before any entries yet exist.

Finally, you should make sure that the database definition contains the index definitions you want:

```
index {<attrlist> | default} [pres,eq,approx,sub,none]
```

For example, to index the `cn`, `sn`, `uid` and `objectclass` attributes, the following `index` directives could be used:

```
index cn,sn,uid
index objectClass pres,eq
```

See The slapd Configuration File section for more details on this option. Once you have configured things to your liking, start up slapd, connect with your LDAP client, and start adding entries. For example, to add an organization entry and an organizational role entry using the *ldapadd* tool, you could create an LDIF file called `entries.ldif` with the contents:

```
# Organization for Example Corporation
dn: dc=example,dc=com
objectClass: dcObject
objectClass: organization
dc: example
o: Example Corporation
description: The Example Corporation

# Organizational Role for Directory Manager
dn: cn=Manager,dc=example,dc=com
objectClass: organizationalRole
cn: Manager
description: Directory Manager
```

and then use a command like this to actually create the entry:

```
ldapadd -f entries.ldif -x -D "cn=Manager,dc=example,dc=com" -w secret
```

The above command assumes settings provided in the above examples.

7.2. Creating a database off-line

The second method of database creation is to do it off-line, using the slapd database tools described below. This method is best if you have many thousands of entries to create, which would take an unacceptably long time to add using the LDAP method described above. These tools read the slapd configuration file and an input file containing a text representation of the entries to add. For database types which support the tools, they produce the database files directly (otherwise you must use the on-line method above). There are several important configuration options you will want to be sure and set in the config file database definition first:

```
suffix <dn>
```

As described in the General Database Directives section, this option defines which entries are to be held by this database. You should set this to the DN of the root of the subtree you are trying to create. For example:

```
suffix "dc=example,dc=com"
```

You should be sure to specify a directory where the index files should be created:

```
directory <directory>
```

For example:

```
directory /usr/local/var/openldap-ldbm
```

Finally, you need to specify which indexes you want to build. This is done by one or more index options.

```
index {<attrlist> | default} [pres,eq,approx,sub,none]
```

For example:

```
index cn,sn,uid pres,eq,approx
index objectClass eq
```

This would create presence, equality and approximate indexes for the `cn`, `sn`, and `uid` attributes and an

equality index for the `objectClass` attribute. See the configuration file section for more information on this option.

7.2.1. The `slapadd` program

Once you've configured things to your liking, you create the primary database and associated indexes by running the *slapadd*(8) program:

```
slapadd -l <inputfile> -f <slapdconfigfile>
        [-d <debuglevel>] [-n <integer>|-b <suffix>]
```

The arguments have the following meanings:

```
-l <inputfile>
```

Specifies the LDIF input file containing the entries to add in text form (described below in the The LDIF text entry format section).

```
-f <slapdconfigfile>
```

Specifies the slapd configuration file that tells where to create the indexes, what indexes to create, etc.

```
-d <debuglevel>
```

Turn on debugging, as specified by `<debuglevel>`. The debug levels are the same as for slapd. See the Command-Line Options section in Running slapd.

```
-n <databasenumber>
```

An optional argument that specifies which database to modify. The first database listed in the configuration file is `1`, the second `2`, etc. By default, the first ldbm database in the configuration file is used. Should not be used in conjunction with `-b`.

```
-b <suffix>
```

An optional argument that specifies which database to modify. The provided suffix is matched against a database `suffix` directive to determine the database number. Should not be used in conjunction with `-n`.

7.2.2. The `slapindex` program

Sometimes it may be necessary to regenerate indices (such as after modifying *slapd.conf*(5)). This is possible using the *slapindex*(8) program. *slapindex* is invoked like this

```
slapindex -f <slapdconfigfile>
        [-d <debuglevel>] [-n <databasenumber>|-b <suffix>]
```

Where the `-f`, `-d`, `-n` and `-b` options are the same as for the *slapadd*(1) program. *slapindex* rebuilds all indices based upon the current database contents.

7.2.3. The `slapcat` program

The `slapcat` program is used to dump the database to an LDIF file. This can be useful when you want to make a human-readable backup of your database or when you want to edit your database off-line. The

program is invoked like this:

```
slapcat -l <filename> -f <slapdconfigfile>
        [-d <debuglevel>] [-n <databasenumber>|-b <suffix>]
```

where `-n` or `-b` is used to select the database in the *slapd.conf*(5) specified using `-f`. The corresponding LDIF output is written to standard output or to the file specified using the `-l` option.

7.3. The LDIF text entry format

The LDAP Data Interchange Format (LDIF) is used to represent LDAP entries in a simple text format. This section provides a brief description of the LDIF entry format which complements *ldif*(5) and the technical specification RFC2849.

The basic form of an entry is:

```
# comment
dn: <distinguished name>
<attrdesc>: <attrvalue>
<attrdesc>: <attrvalue>

...
```

Lines starting with a '`#`' character are comments. An attribute description may be a simple attribute type like `cn` or `objectClass` or `1.2.3` (an OID associated with an attribute type) or may include options such as `cn;lang_en_US` or `userCertificate;binary`.

A line may be continued by starting the next line with a *single* space or tab character. For example:

```
dn: cn=Barbara J Jensen,dc=example,dc=
 com
cn: Barbara J
  Jensen
```

is equivalent to:

```
dn: cn=Barbara J Jensen,dc=example,dc=com
cn: Barbara J Jensen
```

Multiple attribute values are specified on separate lines. e.g.,

```
cn: Barbara J Jensen
cn: Babs Jensen
```

If an `<attrvalue>` contains non-printing characters or begins with a space, a colon ('`:`'), or a less than ('<'), the `<attrdesc>` is followed by a double colon and the base64 encoding of the value. For example, the value "` begins with a space`" would be encoded like this:

```
cn:: IGJlZ2lucyB3aXRoIGEgc3BhY2U=
```

You can also specify a URL containing the attribute value. For example, the following specifies the `jpegPhoto` value should be obtained from the file `/path/to/file.jpeg`.

```
cn:< file:///path/to/file.jpeg
```

Multiple entries within the same LDIF file are separated by blank lines. Here's an example of an LDIF file containing three entries.

```
# Barbara's Entry
dn: cn=Barbara J Jensen,dc=example,dc=com
cn: Barbara J Jensen
cn: Babs Jensen
objectClass: person
sn: Jensen

# Bjorn's Entry
dn: cn=Bjorn J Jensen,dc=example,dc=com
cn: Bjorn J Jensen
cn: Bjorn Jensen
objectClass: person
sn: Jensen
# Base64 encoded JPEG photo
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
 A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
 ERXRTc4UG1RV19iZ2hnPk1xeXBkeFxlZ2P/2wBDARESEhgVG

# Jennifer's Entry
dn: cn=Jennifer J Jensen,dc=example,dc=com
cn: Jennifer J Jensen
cn: Jennifer Jensen
objectClass: person
sn: Jensen
# JPEG photo from file
jpegPhoto:< file:///path/to/file.jpeg
```

Notice that the `jpegPhoto` in Bjorn's entry is base 64 encoded and the `jpegPhoto` in Jennifer's entry is obtained from the location indicated by the URL.

Note: Trailing spaces are not trimmed from values in an LDIF file. Nor are multiple internal spaces compressed. If you don't want them in your data, don't put them there.

---

8. Schema Specification

This chapter describes how to extend the schema used by *slapd*(8). The first section, Distributed Schema Files details optional schema definitions provided in the distribution and where to obtain other definitions. The second section, Extending Schema, details how to define new schema items.

8.1. Distributed Schema Files

OpenLDAP is distributed with a set of schema specifications for your use. Each set is defined in a file suitable for inclusion (using the `include` directive) in your *slapd.conf*(5) file. These schema files are normally installed in the `/usr/local/etc/openldap/schema` directory.

Table 6.1: Provided Schema Specifications

File	Description
`core.schema`	OpenLDAP *core* (required)
`cosine.schema`	Cosine and Internet X.500 (useful)
`inetorgperson.schema`	InetOrgPerson (useful)
`misc.schema`	Assorted (experimental)
`nadf.schema`	North American Directory Forum (FYI)
`nis.schema`	Network Information Services (FYI)
`openldap.schema`	OpenLDAP Project (experimental)

To use any of these schema files, you only need to include the the desired file in the global definitions portion of your *slapd.conf*(5) file. For example:

```
# include schema
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
```

Additional files may be available. Please consult the OpenLDAP FAQ (http://www.openldap.org/faq/).

Note: You should not modify any of the schema items defined in provided files.

8.2. Extending Schema

Schema used by *slapd*(8) may be extended to support additional syntaxes, matching rules, attribute types, and object classes. This chapter details how to add attribute types and object classes using the syntaxes and matching rules already supported by slapd. slapd can also be extended to support additional syntaxes and matching rules, but this requires some programming and hence is not discussed here.

There are five steps to defining new schema:

1. obtain Object Identifer
2. choose a name prefix
3. create local schema file
4. define custom attribute types (if necessary)
5. define custom object classes

8.2.1. Object Identifiers

Each schema element is identified by a globally unique Object Identifier (OID). OIDs are also used to identify other objects. They are commonly found in protocols described by ASN.1. In particular, they are heavily used by the Simple Network Management Protocol (SNMP). As OIDs are hierarchical, your organization can obtain one OID and branch it as needed. For example, if your organization were assigned OID `1.1`, you could branch the tree as follows:

Table 6.2: Example OID hierarchy

OID	Assignment
1.1	Organization's OID
1.1.1	SNMP Elements
1.1.2	LDAP Elements
1.1.2.1	AttributeTypes
1.1.2.1.1	myAttribute
1.1.2.2	ObjectClasses
1.1.2.2.1	myObjectClass

You are, of course, free to design a hierarchy suitable to your organizational needs under your organization's OID. No matter what hierarchy you choose, you should maintain a registry of assignments you make. This can be a simple flat file or a something more sophisticated such as the *OpenLDAP OID Registry* (http://www.openldap.org/faq/index.cgi?file=197).

For more information about Object Identifers (and a listing service) see http://www.alvestrand.no/harald/objectid/.

> *Under no circumstances should you use a fictious OID!*

To obtain a fully registered OID at *no cost*, apply for an OID under Internet Assigned Numbers Authority (IANA) maintained *Private Enterprise* arch. Any private enterprise (organization) may request an OID to be assigned under this arch. Just fill out the IANA form at http://www.iana.org/cgi-bin/enterprise.pl and your official OID will be sent to you usually within a few days. Your base OID will be something like `1.3.6.1.4.1.X` were `X` is an integer.

Note: Don't let the "MIB/SNMP" statement on the IANA page confuse you. OIDs obtained using this form may be used for any purpose including identifying LDAP schema elements.

8.2.2. Name Prefix

In addition to assigning a unique object identifier to each schema element, you should provide a least one textual name for each element. The name should be both descriptive and not likely to clash with names of other schema elements. In particular, any name you choose should not clash with present or future Standard Track names.

To reduce (but not eliminate) the potential for name clashes, the convention is to prefix names of non-Standard Track with a few letters to localize the changes to your organization. The smaller the organization, the longer your prefix should be.

In the examples below, we have choosen a short prefix '`my`' (to save space). Such a short prefix would only be suitable for a very large, global organization. For a small, local organization, we recommend something like '`deFirm`' (German company) or '`comExample`' (elements associated with organization

associated with `example.com`).

8.2.3. Local schema file

The `objectclass` and `attributeTypes` configuration file directives can be used to define schema rules on entries in the directory. It is customary to create a file to contain definitions of your custom schema items. We recommend you create a file `local.schema` in `/usr/local/etc/openldap/schema/local.schema` and then include this file in your *slapd.conf*(5) file immediately after other schema `include` directives.

```
# include schema
include /usr/local/etc/openldap/schema/core.schema
include /usr/local/etc/openldap/schema/cosine.schema
include /usr/local/etc/openldap/schema/inetorgperson.schema
# include local schema
include /usr/local/etc/openldap/schema/local.schema
```

8.2.4. Attribute Type Specification

The *attributetype* directive is used to define a new attribute type. The directive uses the same Attribute Type Description (as defined in RFC2252) used by the attributeTypes attribute found in the subschema subentry, e.g.:

```
attributetype <RFC2252 Attribute Type Description>
```

where Attribute Type Description is defined by the following BNF:

```
AttributeTypeDescription = "(" whsp
      numericoid whsp              ; AttributeType identifier
    [ "NAME" qdescrs ]             ; name used in AttributeType
    [ "DESC" qdstring ]            ; description
    [ "OBSOLETE" whsp ]
    [ "SUP" woid ]                 ; derived from this other
                                   ; AttributeType
    [ "EQUALITY" woid              ; Matching Rule name
    [ "ORDERING" woid              ; Matching Rule name
    [ "SUBSTR" woid ]              ; Matching Rule name
    [ "SYNTAX" whsp noidlen whsp ] ; see section 4.3
    [ "SINGLE-VALUE" whsp ]        ; default multi-valued
    [ "COLLECTIVE" whsp ]          ; default not collective
    [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
    [ "USAGE" whsp AttributeUsage ]; default userApplications
    whsp ")"

AttributeUsage =
    "userApplications"     /
    "directoryOperation"   /
    "distributedOperation" / ; DSA-shared
    "dSAOperation"          ; DSA-specific, value depends on server
```

where whsp is a space (' '), numericoid is a globally unique OID in numeric form (e.g. `1.2.3`), qdescrs is one or more names, woid is either the name or OID, and noidlen is an optional length specifier (e.g `{10}`).

For example, the attribute types `name` and `cn` are defined in `core.schema` as:

```
attributeType ( 2.5.4.41 NAME 'name'
```

```
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
attributeType ( 2.5.4.3 NAME
        ( 'cn' $ 'commonName' ) SUP name )
```

Notice that each defines the attribute's OID and descriptive names. Each name is an alias for the OID. *slapd*(8) returns the first listed name when returning results.

The first attribute, `name`, has a syntax of `directoryString` (a UTF-8 encoded Unicode string) with a recommend maximun length. Note that syntaxes are specified by OID. In addition, the equality and substring matching uses case ignore rules. Below are tables listing commonly used supported syntax and matching rules.

Table 6.3: Supported Syntaxes

Name	**OID**	**Description**
`binary`	`1.3.6.1.4.1.1466.115.121.1.5`	BER/DER data
`boolean`	`1.3.6.1.4.1.1466.115.121.1.7`	boolean value
`distinguishedName`	`1.3.6.1.4.1.1466.115.121.1.12`	DN
`directoryString`	`1.3.6.1.4.1.1466.115.121.1.15`	UTF-8 string
`IA5String`	`1.3.6.1.4.1.1466.115.121.1.26`	ASCII string
`Integer`	`1.3.6.1.4.1.1466.115.121.1.27`	integer
`Name and Optional UID`	`1.3.6.1.4.1.1466.115.121.1.34`	DN plus UID
`Numeric String`	`1.3.6.1.4.1.1466.115.121.1.36`	numeric string
`OID`	`1.3.6.1.4.1.1466.115.121.1.38`	object identifier
`Octet String`	`1.3.6.1.4.1.1466.115.121.1.40`	arbitary octets
`Printable String`	`1.3.6.1.4.1.1466.115.121.1.44`	printable string

Table 6.4: Supported Matching Rules

Name	Type	Description
booleanMatch	equality	boolean
objectIdentiferMatch	equality	OID
distinguishedNameMatch	equality	DN
uniqueMemberMatch	equality	DN with optional UID
numericStringMatch	equality	numerical
numericStringOrderingMatch	ordering	numerical
numericStringSubstringsMatch	substrings	numerical
caseIgnoreMatch	equality	case insensitive, space insensitive
caseIgnoreOrderingMatch	ordering	case insensitive, space insensitive
caseIgnoreSubstringsMatch	substrings	case insensitive, space insensitive
caseExactMatch	equality	case sensitive, space insensitive
caseExactOrderingMatch	ordering	case sensitive, space insensitive
caseExactSubstringsMatch	substrings	case sensitive, space insensitive
caseIgnoreIA5Match	equality	case insensitive, space insensitive
caseIgnoreIA5OrderingMatch	ordering	case insensitive, space insensitive
caseIgnoreIA5SubstringsMatch	substrings	case insensitive, space insensitive
caseExactIA5Match	equality	case sensitive, space insensitive
caseExactIA5OrderingMatch	ordering	case sensitive, space insensitive
caseExactIA5SubstringsMatch	substrings	case sensitive, space insensitive

The second attribute, `cn`, is a subtype of `name` hence it inherits the syntax, matching rules, and usage of `name`. `commonName` is an alternative name.

Neither attribute is restricted to a single value and both are meant for usage by user applications. You likely won't need to specify other parameters such as `OBSOLETE`.

The following subsections provide a couple of examples.

8.2.4.1. myUniqueName

Many organizations maintain a single unique name for each user. Though one could use `displayName` (RFC2798), this attribute is really meant to be controlled by the user, not the organization. We could just copy the definition of `displayName` from `inetorgperson.schema` and replace the OID, name, and description, e.g:

```
attributetype ( 1.1.2.1.1 NAME 'myUniqueName'
        DESC 'unique name with my organization'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
        SINGLE-VALUE )
```

However, if we want this name to be included in `name` assertions [e.g. `(name=*Jane*)`], the attribute could alternatively be defined as a subtype of `name`, e.g.:

```
attributetype ( 1.1.2.1.1 NAME 'myUniqueName'
        DESC 'unique name with my organization'
        SUP name )
```

8.2.4.2. myPhoto

Many organizations maintain a photo of each each user. A `myPhoto` attribute type could be defined to hold a photo. Of course, one could use just use `jpegPhoto` (RFC2798) (or a subtype) to hold the photo. However, you can only do this if the photo is in *JPEG File Interchange Format*. Alternatively, an attribute type which uses the *Octet String* syntax can be defined, e.g.:

```
attributetype ( 1.1.2.1.2 NAME 'myPhoto'
        DESC 'a photo (application defined format)'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.40
        SINGLE-VALUE )
```

As noted in the description, LDAP has no knowledge of the format of the photo. It's assumed that all applications accessing this attribute agree on the handling of values.

If you wanted to support multiple photo formats, you could define a separate attribute type for each format, prefix the photo with some typing information, or describe the value using ASN.1 and use the `;binary` transfer option.

Another alternative is for the attribute to hold a URI pointing to the photo. You can model such an attribute after `labeledURI` (RFC2079).

8.2.5. Object Class Specification

The *objectclasses* directive is used to define a new object class. The directive uses the same Object Class Description (as defined in RFC2252) used by the objectClasses attribute found in the subschema subentry, e.g.:

```
objectclass <RFC2252 Object Class Description>
```

where Object Class Description is defined by the following BNF:

```
ObjectClassDescription = "(" whsp
        numericoid whsp      ; ObjectClass identifier
        [ "NAME" qdescrs ]
        [ "DESC" qdstring ]
        [ "OBSOLETE" whsp ]
        [ "SUP" oids ]       ; Superior ObjectClasses
        [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                ; default structural
        [ "MUST" oids ]      ; AttributeTypes
        [ "MAY" oids ]       ; AttributeTypes
        whsp ")"
```

where whsp is a space (' '), numericoid is a globally unique OID in numeric form (e.g. `1.2.3`), qdescrs is one or more names, and oids is one or more names and/or OIDs.

8.2.5.1. myPhotoObject

To define an *auxiliary* object class which allows myPhoto to be added to any existing entry.

```
objectclass ( 1.1.2.2.1 NAME 'myPhotoObject'
        DESC 'mixin myPhoto'
        AUXILIARY
        MAY myPhoto )
```

8.2.5.2. myPerson

If your organization would like have a private *structural* object class to instantiate users, you can subclass one of the existing person classes, such as `inetOrgPerson` (RFC2798), and add any additional attributes which you desire.

```
objectclass ( 1.1.2.2.2 NAME 'myPerson'
        DESC 'my person'
        SUP inetOrgPerson
        MUST ( 'myUniqueName' $ 'givenName' )
        MAY 'myPhoto' )
```

The object class inherits the required/allowed attribute types of `inetOrgPerson` but requires `myUniqueName` and `givenName` and allows `myPhoto`.

---

9. Constructing a Distributed Directory Service

For many sites, running one or more *slapd*(8) that hold an entire subtree of data is sufficient. But often it is desirable to have one *slapd* refer to other directory services for a certain part of the tree (which may or may not be running *slapd*).

slapd supports *subordinate* and *superior* knowledge information.

9.1. Subordinate Knowledge Information

Subordinate knowledge information may be provided to delegate a subtree. Subordinate knowledge information is maintained in the directory as a special *referral* object at the delegate point. The referral object acts as a delegation point, gluing two services together. This mechanism allows for hierarchical directory services to to be constructed.

A referral object has a structural object class of `referral` and has the same Distinguished Name as the delegated subtree. Generally, the referral object will also provide the auxiliary object class `extensibleObject`. This allows the entry to contain appropriate Relative Distinguished Name values. This is best demonstrated by example.

If the server `a.example.net` holds `dc=example,dc=net` and wished to delegate the subtree `ou=subtree,dc=example,dc=net` to another server `b.example.net`, the following named referral object would be added to `a.example.net`:

```
dn: dc=subtree,dc=example,dc=net
objectClass: referral
```

```
objectClass: extensibleObject
dc: subtree
ref: ldap://b.example.net/dc=subtree,dc=example,dc=net
```

The server uses this information to generate referrals and search continuations to subordinate servers.

For those familiar with X.500, a *named referral* object is similar to an X.500 knowledge reference held in a *subr* DSE.

9.2. Superior Knowledge Information

Superior knowledge information may be specified using the `referral` directive. The value is a list of URIs referring to superior directory services. For servers without immediate superiors, such as for `a.example.net` in the example above, the server can be configured to use directory service with *global knowledge*, such as the *OpenLDAP Root Service* (http://www.openldap.org/faq/index.cgi?file=393).

```
referral        ldap://root.openldap.org/
```

However, as `a.example.net` is the *immediate superior* to `b.example.net`, *b.example.net* would be configured as follows:

```
referral        ldap://a.example.net/
```

The server uses this information to generate referrals to operations acting upon operations not within or subordinate to any of the naming contexts held by the server.

For those familiar with X.500, this use of the `ref` attribute is similar to an X.500 knowledge reference held in a *Supr* DSE.

9.3. The ManageDsaIT Control

Adding, modifying, and deleting referral objects is generally done using *ldapmodify*(1) or similar tools which support the ManageDsaIT control. The ManageDsaIT control informs the server that you intend to manage the referral object as a regular entry. This keeps the server from sending a referral result for requests which interrogate or update referral objects. The -M option of *ldapmodify*(1) (and other tools) enables ManageDsaIT. For example:

```
ldapmodify -M -f referral.ldif -x -D "cn=Manager,dc=example,dc=net" -W
```

or with *ldapsearch*(1):

```
ldapsearch -M -b "dc=example,dc=net" -x "(objectclass=referral)" '*' ref
```

Note: the `ref` attribute is operational and must be explicitly requested when desired in search results.

---

10. Replication with slurpd

In certain configurations, a single *slapd*(8) instance may be insufficient to handle the number of clients requiring directory service via LDAP. It may become necessary to run more than one slapd instance. At many sites, for instance, there are multiple slapd servers: one master and one or more slaves. DNS can be setup such that a lookup of `ldap.example.com` returns the IP addresses of these servers, distributing the load among them (or just the slaves). This master/slave arrangement provides a simple and effective way to increase capacity, availability and reliability.

slurpd(8) provides the capability for a master slapd to propagate changes to slave slapd instances, implementing the master/slave replication scheme described above. slurpd runs on the same host as the master slapd instance.

10.1. Overview

slurpd(8) provides replication services "in band". That is, it uses the LDAP protocol to update a slave database from the master. Perhaps the easiest way to illustrate this is with an example. In this example, we trace the propagation of an LDAP modify operation from its initiation by the LDAP client to its distribution to the slave slapd instance.

Sample replication scenario:

1. The LDAP client submits an LDAP modify operation to the slave slapd.
2. The slave slapd returns a referral to the LDAP client referring the client to the master slapd.
3. The LDAP client submits the LDAP modify operation to the master slapd.
4. The master slapd performs the modify operation, writes out the change to its replication log file and returns a success code to the client.
5. The slurpd process notices that a new entry has been appended to the replication log file, reads the replication log entry, and sends the change to the slave slapd via LDAP.
6. The slave slapd performs the modify operation and returns a success code to the slurpd process.

10.2. Replication Logs

When slapd is configured to generate a replication logfile, it writes out a file containing LDIF change records. The replication log gives the replication site(s), a timestamp, the DN of the entry being modified, and a series of lines which specify the changes to make. In the example below, Barbara (`uid=bjensen`) has replaced the `description` value. The change is to be propagated to the slapd instance running on `slave.example.net` Changes to various operational attributes, such as `modifiersName` and `modifyTimestamp`, are included in the change record and will be propagated to the slave slapd.

```
replica: slave.example.com:389
time: 809618633
dn: uid=bjensen,dc=example,dc=com
changetype: modify
replace: multiLineDescription
description: A dreamer...
-
replace: modifiersName
modifiersName: uid=bjensen,dc=example,dc=com
-
replace: modifyTimestamp
modifyTimestamp: 200008050073308Z
-
```

The modifications to `modifiersName` and `modifyTimestamp` operational attributes were added by the master *slapd*.

10.3. Command-Line Options

This section details commonly used *slurpd*(8) command-line options.

```
-d <level> | ?
```

This option sets the slurpd debug level to `<level>`. When level is a `?' character, the various debugging levels are printed and slapd exits, regardless of any other options you give it. Current debugging levels (a subset of slapd's debugging levels) are

Table 10.1: Debugging Levels

Level	Description
4	heavy trace debugging
64	configuration file processing
65535	enable all debugging

Debugging levels are additive. That is, if you want heavy trace debugging and want to watch the config file being processed, you would set level to the sum of those two levels (in this case, 68).

```
-f <filename>
```

This option specifies an alternate slapd configuration file. Slurpd does not have its own configuration file. Instead, all configuration information is read from the slapd configuration file.

```
-r <filename>
```

This option specifies an alternate slapd replication log file. Under normal circumstances, slurpd reads the name of the slapd replication log file from the slapd configuration file. However, you can override this with the -r flag, to cause slurpd to process a different replication log file. See the Advanced slurpd Operation section for a discussion of how you might use this option.

```
-o
```

Operate in "one-shot" mode. Under normal circumstances, when slurpd finishes processing a replication log, it remains active and periodically checks to see if new entries have been added to the replication log. In one-shot mode, by comparison, slurpd processes a replication log and exits immediately. If the -o option is given, the replication log file must be explicitly specified with the -r option. See the One-shot mode and reject files section for a discussion of this mode.

```
-t <directory>
```

Specify an alternate directory for slurpd's temporary copies of replication logs. The default location is /usr/tmp.

10.4. Configuring slurpd and a slave slapd instance

To bring up a replica slapd instance, you must configure the master and slave slapd instances for replication, then shut down the master slapd so you can copy the database. Finally, you bring up the master slapd instance, the slave slapd instance, and the slurpd instance. These steps are detailed in the following sections. You can set up as many slave slapd instances as you wish.

10.4.1. Set up the master *slapd*

The following section assumes you have a properly working *slapd*(8) instance. To configure your working *slapd*(8) server as a replication master, you need to make the following changes to your *slapd.conf*(5).

1. Add a `replica` directive for each replica. The `binddn=` parameter should match the `updatedn` option in the corresponding slave slapd configuration file, and should name an entry with write permission to the slave database (e.g., an entry listed as `rootdn`, or allowed access via `access` directives in the slave slapd configuration file).
2. Add a `replogfile` directive, which tells slapd where to log changes. This file will be read by slurpd.

10.4.2. Set up the slave *slapd*

Install the slapd software on the host which is to be the slave slapd server. The configuration of the slave server should be identical to that of the master, with the following exceptions:

1. Do not include a `replica` directive. While it is possible to create "chains" of replicas, in most cases this is inappropriate.
2. Do not include a `replogfile` directive.
3. Do include an `updatedn` line. The DN given should match the DN given in the `binddn=` parameter of the corresponding `replica=` directive in the master slapd config file.
4. Make sure the DN given in the `updatedn` directive has permission to write the database (e.g., it is listed as `rootdn` or is allowed `access` by one or more access directives).
5. Use the `updateref` directive to define the URL the slave should return if an update request is received.

10.4.3. Shut down the master *slapd*

In order to ensure that the slave starts with an exact copy of the master's data, you must shut down the master slapd. Do this by sending the master slapd process an interrupt signal with `kill -INT <pid>`, where `<pid>` is the process-id of the master slapd process.

If you like, you may restart the master slapd in read-only mode while you are replicating the database. During this time, the master slapd will return an "unwilling to perform" error to clients that attempt to modify data.

10.4.4. Copy the master slapd's database to the slave

Copy the master's database(s) to the slave. For an LDBM-based database, you must copy all database files located in the database `directory` specified in *slapd.conf*(5). Database files will have a different suffix depending on the underlying database package used. The current possibilities are

Table 10.2: Database File Suffixes

Suffix	Database
dbb	Berkeley DB B-tree backend
dbh	Berkeley DB hash backend
gdbm	GNU DBM backend

In general, you should copy each file found in the database directory unless you know it is not used by *slapd*(8).

Note: The copy process assumes homogeneous servers with identically configured OpenLDAP installations.

10.4.5. Configure the master slapd for replication

To configure slapd to generate a replication logfile, you add a " replica" configuration option to the master slapd's config file. For example, if we wish to propagate changes to the slapd instance running on host slave.example.com:

```
replica host=slave.example.com:389
        binddn="cn=Replicator,dc=example,dc=com"
        bindmethod=simple credentials=secret
```

In this example, changes will be sent to port 389 (the standard LDAP port) on host slave.example.com. The slurpd process will bind to the slave slapd as "cn=Replicator,dc=example,dc=com" using simple authentication with password "secret". Note that the DN given by the binddn= directive must exist in the slave slapd's database (or be the rootdn specified in the slapd config file) in order for the bind operation to succeed. The DN should also be listed as the updatedn for the database in the slave's slapd.conf(5).

Note: The use of strong authentication and transport security is highly recommended.

10.4.6. Restart the master slapd and start the slave slapd

Restart the master slapd process. To check that it is generating replication logs, perform a modification of any entry in the database, and check that data has been written to the log file.

10.4.7. Start slurpd

Start the slurpd process. Slurpd should immediately send the test modification you made to the slave slapd. Watch the slave slapd's logfile to be sure that the modification was sent.

```
slurpd -f <masterslapdconfigfile>
```

10.5. Advanced slurpd Operation

10.5.1. Replication errors

When slurpd propagates a change to a slave slapd and receives an error return code, it writes the reason for the error and the replication record to a reject file. The reject file is located in the same directory as the per-replica replication logfile, and has the same name, but with the string "`.rej`" appended. For example, for a replica running on host `slave.example.com`, port 389, the reject file, if it exists, will be named

```
/usr/local/var/openldap/replog.slave.example.com:389.rej
```

A sample rejection log entry follows:

```
ERROR: No such attribute
replica: slave.example.com:389
time: 809618633
dn: uid=bjensen,dc=example,dc=com
changetype: modify
replace: description
description: A dreamer...
-
replace: modifiersName
modifiersName: uid=bjensen,dc=example,dc=com
-
replace: modifyTimestamp
modifyTimestamp: 20000805073308Z
-
```

Note that this is precisely the same format as the original replication log entry, but with an `ERROR` line prepended to the entry.

10.5.2. One-shot mode and reject files

It is possible to use slurpd to process a rejection log with its "one-shot mode." In normal operation, slurpd watches for more replication records to be appended to the replication log file. In one-shot mode, by contrast, slurpd processes a single log file and exits. Slurpd ignores `ERROR` lines at the beginning of replication log entries, so it's not necessary to edit them out before feeding it the rejection log.

To use one-shot mode, specify the name of the rejection log on the command line as the argument to the -r flag, and specify one-shot mode with the -o flag. For example, to process the rejection log file `/usr/local/var/openldap/replog.slave.example.com:389` and exit, use the command

```
slurpd -r /usr/tmp/replog.slave.example.com:389 -o
```

A. Generic configure Instructions

```
Basic Installation
==================

   These are generic installation instructions.
```

```
   The `configure' shell script attempts to guess correct values for
various system-dependent variables used during compilation.  It uses
those values to create a `Makefile' in each directory of the package.
It may also create one or more `.h' files containing system-dependent
definitions.  Finally, it creates a shell script `config.status' that
you can run in the future to recreate the current configuration, a file
`config.cache' that saves the results of its tests to speed up
reconfiguring, and a file `config.log' containing compiler output
(useful mainly for debugging `configure').

   If you need to do unusual things to compile the package, please try
to figure out how `configure' could check whether to do them, and mail
diffs or instructions to the address given in the `README' so they can
be considered for the next release.  If at some point `config.cache'
contains results you don't want to keep, you may remove or edit it.

   The file `configure.in' is used to create `configure' by a program
called `autoconf'.  You only need `configure.in' if you want to change
it or regenerate `configure' using a newer version of `autoconf'.

The simplest way to compile this package is:

  1. `cd' to the directory containing the package's source code and type
     `./configure' to configure the package for your system.  If you're
     using `csh' on an old version of System V, you might need to type
     `sh ./configure' instead to prevent `csh' from trying to execute
     `configure' itself.

     Running `configure' takes awhile.  While running, it prints some
     messages telling which features it is checking for.

  2. Type `make' to compile the package.

  3. Optionally, type `make check' to run any self-tests that come with
     the package.

  4. Type `make install' to install the programs and any data files and
     documentation.

  5. You can remove the program binaries and object files from the
     source code directory by typing `make clean'.  To also remove the
     files that `configure' created (so you can compile the package for
     a different kind of computer), type `make distclean'.  There is
     also a `make maintainer-clean' target, but that is intended mainly
     for the package's developers.  If you use it, you may have to get
     all sorts of other programs in order to regenerate files that came
     with the distribution.

Compilers and Options
=====================

   Some systems require unusual options for compilation or linking that
the `configure' script does not know about.  You can give `configure'
initial values for variables by setting them in the environment.  Using
a Bourne-compatible shell, you can do that on the command line like
this:
     CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure

Or on systems that have the `env' program, you can do it like this:
     env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure

Compiling For Multiple Architectures
====================================
```

```
   You can compile the package for more than one kind of computer at the
same time, by placing the object files for each architecture in their
own directory.  To do this, you must use a version of `make' that
supports the `VPATH' variable, such as GNU `make'.  `cd' to the
directory where you want the object files and executables to go and run
the `configure' script.  `configure' automatically checks for the
source code in the directory that `configure' is in and in `..'.

   If you have to use a `make' that does not supports the `VPATH'
variable, you have to compile the package for one architecture at a time
in the source code directory.  After you have installed the package for
one architecture, use `make distclean' before reconfiguring for another
architecture.

Installation Names
==================

   By default, `make install' will install the package's files in
`/usr/local/bin', `/usr/local/man', etc.  You can specify an
installation prefix other than `/usr/local' by giving `configure' the
option `--prefix=PATH'.

   You can specify separate installation prefixes for
architecture-specific files and architecture-independent files.  If you
give `configure' the option `--exec-prefix=PATH', the package will use
PATH as the prefix for installing programs and libraries.
Documentation and other data files will still use the regular prefix.

   In addition, if you use an unusual directory layout you can give
options like `--bindir=PATH' to specify different values for particular
kinds of files.  Run `configure --help' for a list of the directories
you can set and what kinds of files go in them.

   If the package supports it, you can cause programs to be installed
with an extra prefix or suffix on their names by giving `configure' the
option `--program-prefix=PREFIX' or `--program-suffix=SUFFIX'.

Optional Features
=================

   Some packages pay attention to `--enable-FEATURE' options to
`configure', where FEATURE indicates an optional part of the package.
They may also pay attention to `--with-PACKAGE' options, where PACKAGE
is something like `gnu-as' or `x' (for the X Window System).  The
`README' should mention any `--enable-' and `--with-' options that the
package recognizes.

   For packages that use the X Window System, `configure' can usually
find the X include and library files automatically, but if it doesn't,
you can use the `configure' options `--x-includes=DIR' and
`--x-libraries=DIR' to specify their locations.

Specifying the System Type
==========================

   There may be some features `configure' can not figure out
automatically, but needs to determine by the type of host the package
will run on.  Usually `configure' can figure that out, but if it prints
a message saying it can not guess the host type, give it the
`--host=TYPE' option.  TYPE can either be a short name for the system
type, such as `sun4', or a canonical name with three fields:
     CPU-COMPANY-SYSTEM

See the file `config.sub' for the possible values of each field.  If
`config.sub' isn't included in this package, then this package doesn't
```

```
need to know the host type.

   If you are building compiler tools for cross-compiling, you can also
use the `--target=TYPE' option to select the type of system they will
produce code for and the `--build=TYPE' option to select the type of
system on which you are compiling the package.

Sharing Defaults
================

   If you want to set default values for `configure' scripts to share,
you can create a site shell script called `config.site' that gives
default values for variables like `CC', `cache_file', and `prefix'.
`configure' looks for `PREFIX/share/config.site' if it exists, then
`PREFIX/etc/config.site' if it exists.  Or, you can set the
`CONFIG_SITE' environment variable to the location of the site script.
A warning: not all `configure' scripts look for a site script.

Operation Controls
==================

   `configure' recognizes the following options to control how it
operates.

`--cache-file=FILE'
     Use and save the results of the tests in FILE instead of
     `./config.cache'.  Set FILE to `/dev/null' to disable caching, for
     debugging `configure'.

`--help'
     Print a summary of the options to `configure', and exit.

`--quiet'
`--silent'
`-q'
     Do not print messages saying which checks are being made.  To
     suppress all normal output, redirect it to `/dev/null' (any error
     messages will still be shown).

`--srcdir=DIR'
     Look for the package's source code in directory DIR.  Usually
     `configure' can determine that directory automatically.

`--version'
     Print the version of Autoconf used to generate the `configure'
     script, and exit.

`configure' also accepts some other, not widely useful, options.
```

B. OpenLDAP Software Copyright Notices

B.1. OpenLDAP Copyright Notice

`LICENSE` in the top-level directory of the distribution.

Individual files and/or contributed packages may be copyright by other parties and their use subject to additional restrictions.

This work is derived from the University of Michigan LDAP v3.3 distribution. Information concerning this software is available at:

> http://www.umich.edu/~dirsvcs/ldap/.

This work also contains materials derived from public sources.

Additional Information about OpenLDAP can be obtained at:

> http://www.OpenLDAP.org/

or by sending e-mail to:

> info@OpenLDAP.org

B.2. University of Michigan Copyright Notice

Portions Copyright 1992-1996 Regents of the University of Michigan. All rights reserved.

Redistribution and use in source and binary forms are permitted provided that this notice is preserved and that due credit is given to the University of Michigan at Ann Arbor. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission. This software is provided ``as is'' without express or implied warranty.

C. The OpenLDAP Public License

```
The OpenLDAP Public License
  Version 2.4, 8 December 2000

Redistribution and use of this software and associated documentation
("Software"), with or without modification, are permitted provided
that the following conditions are met:

1. Redistributions of source code must retain copyright statements
and notices.

2. Redistributions in binary form must reproduce applicable copyright
statements and notices, this list of conditions, and the following
disclaimer in the documentation and/or other materials provided
with the distribution.

3. Redistributions must contain a verbatim copy of this document.

4. The names and trademarks of the authors and copyright holders
must not be used in advertising or otherwise to promote the sale,
use or other dealing in this Software without specific, written
```

Home | Catalog